IT UNIVERSITY OF COPENHAGEN

DevOps, Software Evolution and Software Maintenance

COURSE CODE: KSDSESM1KU

MSC IN COMPUTER SCIENCE

DevOps: ITU-MiniTwit

GROUP E — GRL PWR

IT UNIVERSITY OF COPENHAGEN

Name	Email
Amalie Bøgild Sørensen	abso@itu.dk
Andreas Nicolaj Tietgen	anti@itu.dk
Malin Birgitta Linnea Nordqvist	bino@itu.dk
Mille Mei Zhen Loo	milo@itu.dk
My Marie Nordal Jensen	myje@itu.dk
Sarah Cecilie Chytræus Christians	sacc@itu.dk

May 9, 2024

2500 words

Table of Contents

1 System Perspective

2 Process Perspective

hallo [2]

2.1 Monitoring and logging

For monitoring we use Grafana. Our board shows requests duration, errors rate, top 10 unhandled exception endpoints and more (for visualization of the board, see appendix x). We have used the board to get an overview of where to put our focus, ie. which endpoints to improve, which errors to fix, etc. Moreover, we have gained insights into the health of the system and gotten an impression of how the backend and frontend handle the requests.

For logging, we use Grafana and Loki. It seemed obvious to continue our work with Grafana in order to keep the system setup as simple as possible. The logs are divided into Information, Warning, Debug and Error. All logging statements are placed in the controllers, such that we have information about the users' whereabouts in the system. For example, we log when a user logins whether successful or unsuccessful.

3 Lessons Learned

3.1 Lesson 1: Getting Hacked

Just a couple of hours after attending the lecture on security, we got hacked, leading us to experience firsthand how important it is to incorporate security in a CI/CD pipeline.

The first suspicion we got was when we discovered, through our monitoring, that the response time of our server was suddenly very slow. This led us to our Digital Ocean dashboard which showed that the server was using 100% CPU power, which is highly unusual.

From that point the group scoured the server for clues as to what was happening, finding countless calls to `masscan` essentially drowning our server, as well as mysterious installations and what looked like a call to a remote script via a cronjob.

After a few failed attempts to evict the adversary it was decided to destroy the server, as we fortunately had already implemented our Infrastructure as Code, so provisioning and deploying a new server could be done in under half an hour.

After some introspection into our system, we assume that the adversary had gotten access via some ports that we were unaware were exposed. The ports became exposed in an attempt to make the network function between servers in a docker swarm, which seems to override the firewall.

Learning from this, we have worked to close exposed ports from Docker and finding alternative solutions to setting up the network. Another key takeaway is that because we had the necessary monitoring in place, to figure out that the server was being targeted, as well as having our Infrastructure as Code, we were able to detect and react to the attack fairly quickly, giving us only a few hours of downtime.

3.2 Lesson 2: Shift from Vagrant to Ansible-Pulumi

Another lesson we have learned during this project is the importance of choosing the right tools for a project. At the start of the project, we had chosen to provision our VMs with Vagrant, inspired by the exercises from the course. When realizing later in the project that we would have to switch Digital Ocean account at some point due to running out of credit, we had to streamline the setup of our VMs. The choice of Configuration Management tool fell upon Ansible, which was supposed to call the Vagrant files from a config server, provisioning the web and monitoring servers. However, when having more complicated automation and collaboration needs for our project, it turned out that Vagrant was not the right tool for the job. After many hours of attempting to get Vagrant to work with Ansible, we found out that Vagrant saves local metadata to maintain some state, which was making the provisioning from Ansible and the config server fail[1]. Furthermore, Vagrant does not seem to be supported with GitHub Actions - possibly also due to how it handles state. We decided to cut our losses with Vagrant and search for a more suitable tool that could help us write our infrastructure as code. The choice fell upon Pulumi in the end. It was a valuable lesson to see how it made a difference when we took the time to investigate different tools and their properties so we could make an informed decision based on the knowledge of our system's needs.

4 References

[1] DevOps Group e. *Github Issue: Add playbooks in the vagrant provisioning steps*. 2024. URL: https://github.com/devops2024-group-e/itu-minitwit/issues/178.

[2] Gene Kim. *The DevOPS Handbook - How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. It Revolution Press, 2016. ISBN: 9781942788003.

Appendices